Functional requirements

The system will have to have at least a simple set of possible actions in order to work:

(users)

- create user, with:
  - BUCS user name
  - Full name
  - Telephone number
  - Password
- log in
- remember that the user has logged in (sessions), which creates a new attribute:
  - Session ID
- change the details
- log out (erase session)

(books)

- enter a book in the system, with:
  - ISBN
  - Title
  - Authors
  - Edition number
  - Publication date
  - Publisher
  - Subject
- recognize a pre-entered book
- users enter books, and they may enter false information. We should therefore store who has entered the information, and let other users tell us if a book info is incorrect... New attribute:
  - BUCS ID of the person who has entered the book information

(sells and bids)

- create a sell, with:
  - Book condition
  - Reserve price
  - Initial price
  - For how many days to sell the book
- stop selling a book, therefore we have a new attribute:
  - Sell status
- bid, with:
  - Bid price
- bid more

Data organization

Based on the problem description, we can create a first version of data organization:

> **Users table**: BUCS ID, Forename, Surname, Password, Telephone and Session ID
>
> **Books table**: ISBN, BUCS ID of the person who's entered the information, Title, Authors, Edition number, Publication date, Publisher and Subject
>
> **Sells table**: ISBN, Seller's BUCS ID, Book condition, Reserve price, Initial price, Date started, End date, Status, Bidder BUCS ID, Bid price

The users and books tables seem to have no functional dependencies (except some minor possible dependencies between Authors, Publishers and Subjects, since actually most publishers will generally publish books on certain subjects, and the same for authors...), still the sells table obviously has functional dependencies... One first idea would be to divide it in two parts:

> **Sells table**: Sell ID, ISBN, Seller's BUCS ID, Book condition, Reserve price, Initial price, Date started, End date and Status.
>
> **Bids table**: Sell ID the user bid for, bidder's BUCS ID and the price

We should now find the primary keys for out tables:

> **Users table**: we could have multiple candidates, since, at a first sight the telephone number, BUCS ID or even the session ID are some unique values... First of all, session ID is impossible since all logged out users will be marked the same, and even if we did mark every logged out user uniquely the session ID is supposed to change quite often, so is a really bad key! The same argument can also be applied to the telephone number (even though there is the number portability system in the UK). Two different people may have a same name and surname, therefore the ideal key for this table is **BUCS ID**.
>
> **Books table**: the **ISBN** number is already a World standard for uniquely marking every book... Just note that it normally does not indicate the student that has entered this book in our system, still this information is unique in our database and since every book can only be entered by one user we have ISBN -> BUCS ID...
>
> **Sells table**: the ISBN identifies the books, BUCS ID a user, book condition, reserve price, bid start date, etc. are all useless by themselves. We could still say that the combination of BUCS ID and bid start time is probably a good key, still is too long and this reality may be changed by a user that refreshes pages or by some selling robot softwares... The best idea would therefore be to put a new key that does not correspond to any physical reality, that we will call **Sell ID**.
>
> **Bids table**: when a user bids a second time for a same sell, his / her older bid is overwritten. Therefore, the combination of **Sell ID** and **BUCS ID** is sufficient to find a bid. During the creation of the application, this turned out to work pretty well and we did not need any key that is just one attribute.

With those four tables, our relations satisfy BCNF.

In practice...

To make the system secure, we need to be careful about a few issues:

- **Too long inputs**
  Some users may try to create false forms to submit data longer than expected. Even though in MySQL all extra data will bu cut during INSERTs and UPDATEs, it is good to keep this in control from the beginning. In all cases, it will at least save the bandwidth used by the script...

- **Meaningful characters**
  Another point we should not miss is that the output will be in HTML, and HTML uses tags... What if a user inputs data that match special HTML codes? This should definitely be avoided as well

- **XSS (cross-side scripting)**
  Users may also enter some other weird king of data: let's suppose we have a beautiful query like:

```
SELECT * FROM Users_table WHERE
Username = "$username" AND Password = "$password"
```

  It seems OK, doesn't it? Well, it shouldn't! Let's suppose that the user gave as username not a normal name but something weird, like `admin"-`... What effect does it have on our query?

```
SELECT * FROM Users_table WHERE
Username = "admin"-" AND Password = "$password"
```

  Since the – character stands for "the rest is just comments", the `" AND Password = "$password"` gets ignored; so the user is logged in as admin, not matter how complicated the admin password may be! We should therefore also filter out this kind of special characters

- **Viewed database**
  One important point is that even if one can view invisible parts of the database, the information retrieved should be useless. This can be done with the password, and we therefore encrypt the passwords using the PHP md5 command

- **Session ID security**
  The last point is that session IDs should be hardly guessable, therefore not just a simple increment like 1, 2, 3, 4, ... To create session IDs, we therefore use the time() multiplied by a 11-digit random number, that we then transform into base 36, and then encode using md5. This gives us a 32 character long string.

- **Make sure users do not enter really "weird" information**
  That is the last point: we should also check that information entered by the user at least satisfies some criteria, for instance a telephone number is a 10-digit long int, an ISBN number is an int, a date has a certain format, etc. We could also render the system better by sending users some verification e-mails, so we are sure that their e-mail addresses work.

Also, to make the system better, we could have integrated a module that would send and e-mail when one user wins a bid (to the seller and to the winner), which would make the application even more user friendly... But, problem: when testing the "alert the administrator about bad book info" functionality, we've seen that sending e-mails using the PHP on campus is impossible! The idea has then been thrown out, still not forgotten since we mention it here.

Database optimization

When the application is finished, we clearly see that some attributes are read very often:

- The bid price for a given Sell ID and / or BUCS ID (and never the inverse), and this is often (or always) sorted by a descending bid price
- Book information for a given ISBN number (yes, other information is accessed when a user is searching for a book, but a user will probably search for it once and then just bid, and bid again, and again; and at each bid we get the book info based on its ISBN number)
- Sell information for a given BUCS ID, and sometimes by Sell ID (but the second case is quite rare compared to the first one)
- User information by BUCS ID (when profile is accessed, a user wins the bid for an item sold by this user, etc.) and BUCS ID for a given session ID (every time the user changes page)

We can therefore try to create indexes for those things, but first we need to inspect the information MySQL gives us for the tables:

- In the bids table, we have no primary key or index, therefore no predefined index... We have to index the table by Sell ID and by BUCS ID.
- ISBN is the primary key of the books table, therefore also an index: the second point can be ignored.
- The sell table has Sell ID as primary key, therefore is by default indexed by Sell ID: we should tell MySQL to also index by BUCS ID.
- The users table is indexed by BUCS ID (primary key), therefore the creation of an extra index for session ID would be good...

So here is the query to create the tables:

```
DROP TABLE IF EXISTS `st221_project_users`;
CREATE TABLE `st221_project_users` (
  `BUCS_id` varchar(8) NOT NULL,
  `Forename` varchar(32) NOT NULL,
  `Surname` varchar(32) NOT NULL,
  `Password` varchar(32) NOT NULL,
  `Telephone` bigint(40) NOT NULL,
  `Session_id` varchar(32) NOT NULL,
  PRIMARY KEY  (`BUCS_id`),
  INDEX  (`Session_id`)
) TYPE=MyISAM
  COMMENT='Users table... password & session_id use PHP md5 encryption';

DROP TABLE IF EXISTS `st221_project_books`;
CREATE TABLE `st221_project_books` (
  `ISBN` bigint(40) NOT NULL,
  `BUCS_id` varchar(8) NOT NULL REFERENCES `st221_project_users`,
  `Title` varchar(32) NOT NULL,
  `Authors` varchar(32) NOT NULL,
  `Edition_nb` int(8) NOT NULL,
  `Publication_date` varchar(8) NOT NULL,
  `Publisher` varchar(32) NOT NULL,
  `Subject` enum('Biology', 'Economy', 'Mathematics', 'Sociology', 'Computers',
               'Mechanics', 'Physics', 'Literature', 'History', 'Geography',
               'Philosophy') NOT NULL,
  PRIMARY KEY  (`ISBN`)
) TYPE=MyISAM
  COMMENT='Books table... We also store who entered book info';

DROP TABLE IF EXISTS `st221_project_sells`;
CREATE TABLE `st221_project_sells` (
  `Sell_id` int(8) NOT NULL AUTO_INCREMENT,
  `BUCS_id` varchar(8) NOT NULL REFERENCES `st221_project_users`,
  `ISBN` bigint(40) NOT NULL REFERENCES `st221_project_books`,
  `Condition` enum('As good as new', 'Good', 'Medium','Quite bad, still readable',
                   'Really bad: missing / unreadable pages') NOT NULL,
  `Reserve_price` int(8) NOT NULL,
  `Initial_price` int(8) NOT NULL,
  `Date_started` int(11) NOT NULL,
  `Date_ending` int(11) NOT NULL,
  `Status` enum('on sale','removed','sold') NOT NULL,
  PRIMARY KEY  (`Sell_id`),
  INDEX  (`BUCS_id`)
) TYPE=MyISAM
  COMMENT='Sell details... Prices are in pounds * 10';

DROP TABLE IF EXISTS `st221_project_bids`;
CREATE TABLE `st221_project_bids` (
  `Sell_id` int(8) NOT NULL REFERENCES `st221_project_sell_details`,
  `BUCS_id` varchar(8) NOT NULL REFERENCES `st221_project_users`,
  `Bid_price` int(8) NOT NULL,
  INDEX  (`BUCS_id`),
  INDEX  (`Sell_id`)
) TYPE=MyISAM
  COMMENT='Bids table... Prices are in pounds * 10';
```